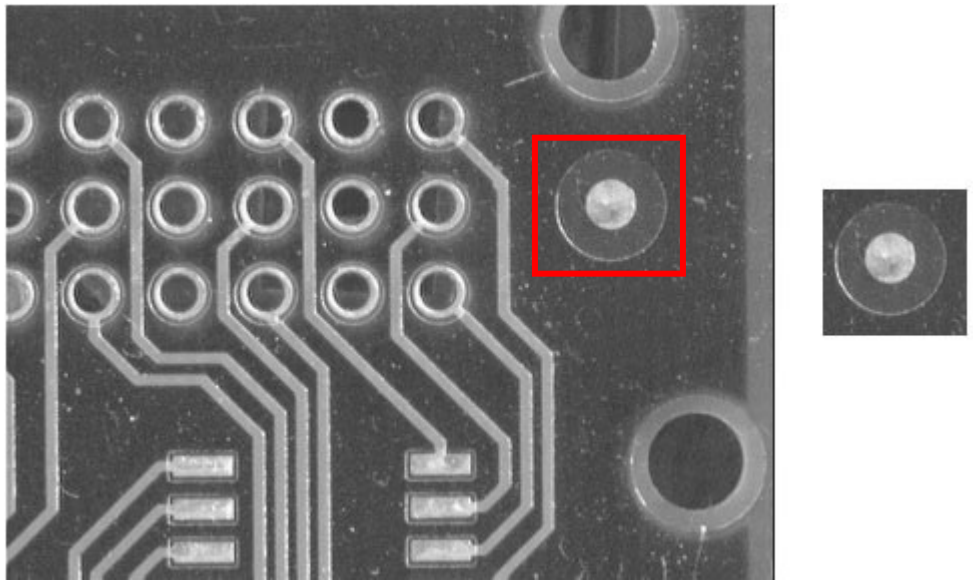
Image Processing and Computer Vision

Prof. Giuseppe Lisanti
giuseppe.lisanti@unibo.it

Instance-level Object Detection

- The *Instance-level Object Detection* problem occurs in countless applications and can be formulated as follows.
  - Given a reference image (aka model image) of a *specific object*, determine whether the object is present or not in the image under analysis (aka target image) and, in case of detection, estimate the pose of the object.



- Depending on the application, the pose may often be given by a translation, a roto-translation or a similarity (roto-translation plus scale)

Instance-level Object Detection

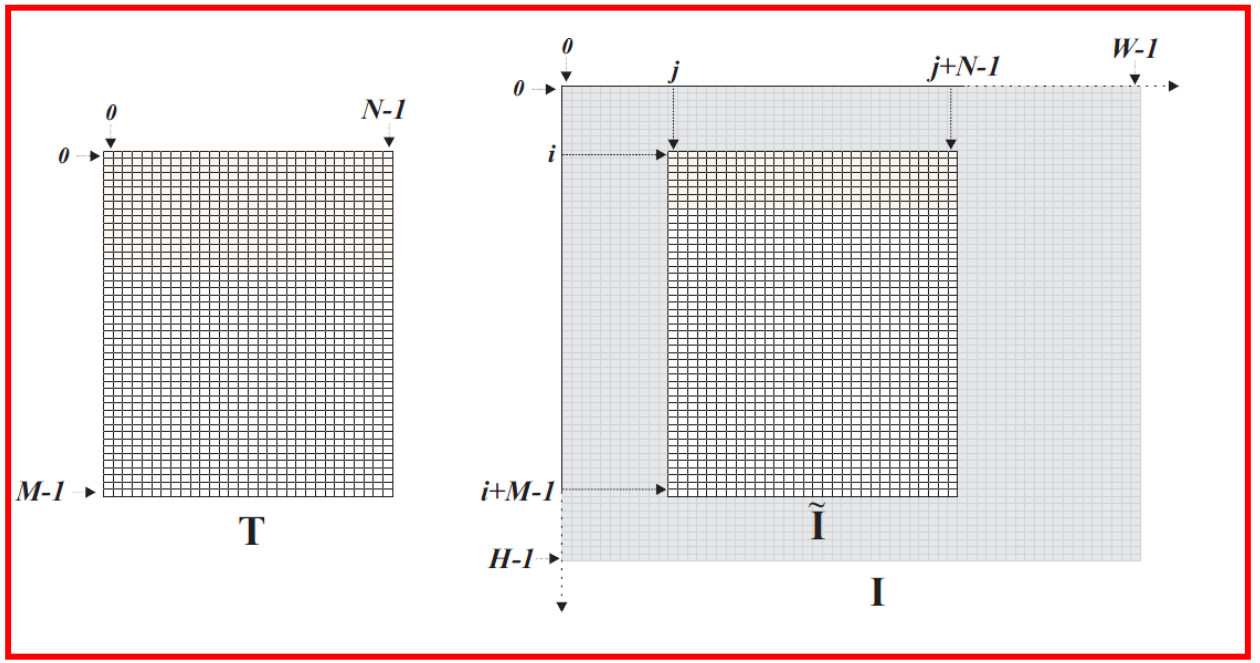
- The problem has a number of diverse facets: the sought object may appear only once or multiple times in the target image, we may be interested in detecting a number of diverse objects, each of which, in turn, may appear once of multiple times.
  - For the sake of simplicity, and without much loss of generality, we will refer mainly to the basic setting: *detecting a single object that may appear once in the target image*. Generalization to the other variants turns out more often than not straightforward.
- Typical nuisances to be dealt with are *intensity changes, occlusions and clutter*.
- Computational efficiency is a major requirement in most practical applications.
- This problem is characterized by a *limited variability* as the assumption deals with the appearance of the object being captured by a single model image (i.e. roughly planar objects or no view-point changes) and the pose is typically either a 2D translation or a 2D roto-translation or a similarity.
- Given the limited variability, the problem can been addressed successfully by classical computer vision techniques, the major applications dealing mainly with the space of industrial vision.

- Instead, *Category-level Object Detection* aims at detecting certain kind of object(s) (e.g. cars, pedestrians..) regardless of their appearance and pose. Due to the high-variability, this problem is addressed by machine/deep learning techniques.

Template Matching

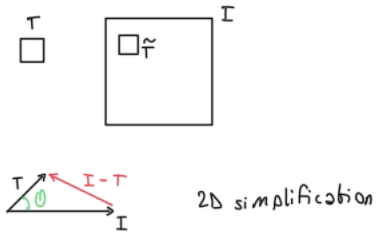
- The model image is slid across the target image to be compared at each position to an equally sized window by means of a suitable (dis)similarity function

(Dis)Similarity Functions



- Both $\tilde{I}(i,j)$, the window at position (i, j) of the target image having the same size as T, as well as T can be thought of as M·N-dimensional vectors (flatten)
- Compute pixel-wise intensities differences: $SSD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - T(m,n)\right)^2$

(Dis)Similarity Functions



- Sum of Absolute Differences: $$SAD(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|I(i+m,j+n) - T(m,n)\right|$$

- Are SSD and SAD invariant to intensity changes? NO, when we should use them?

- Normalised Cross-Correlation: $$NCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)\cdot T(m,n)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)^2}}$$ Similarity!

  - Represents the cosine of the angle between vectors I(i,j) e T $$NCC(i,j) = \frac{\tilde{\mathbf{I}}(i,j)\cdot\mathbf{T}}{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|} = \frac{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|\cdot\cos\theta}{\left\|\tilde{\mathbf{I}}(i,j)\right\|\cdot\|\mathbf{T}\|} = \cos\theta$$

  - Invariant to linear intensity changes $$\tilde{\mathbf{I}}(i,j) = \alpha\cdot\mathbf{T}$$

(Dis)Similarity Functions

- Zero-Mean Normalised Cross-Correlation, Correlation Coefficient

$$\mu(\tilde{I}) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n) \qquad \mu(T) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)$$

- The NCC is computed after subtraction of the means:

Mean of the subimage

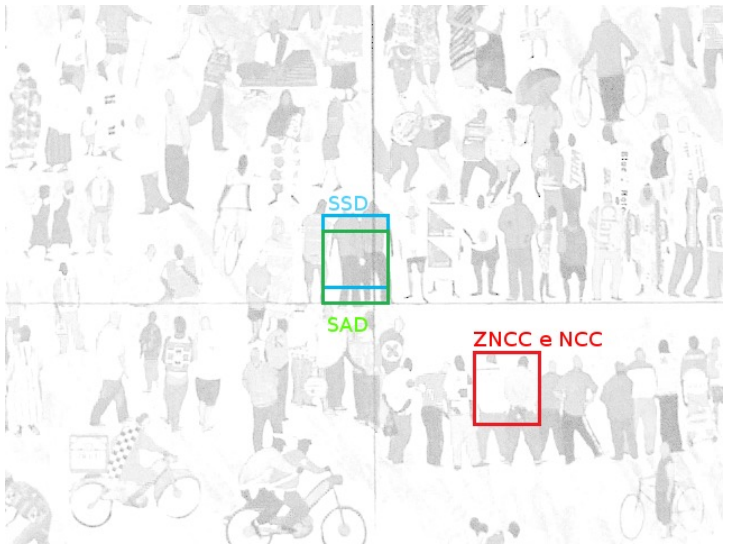
$$I(i+m, j+n) \rightarrow \left(I(i+m, j+n) - \mu(\tilde{I})\right)$$

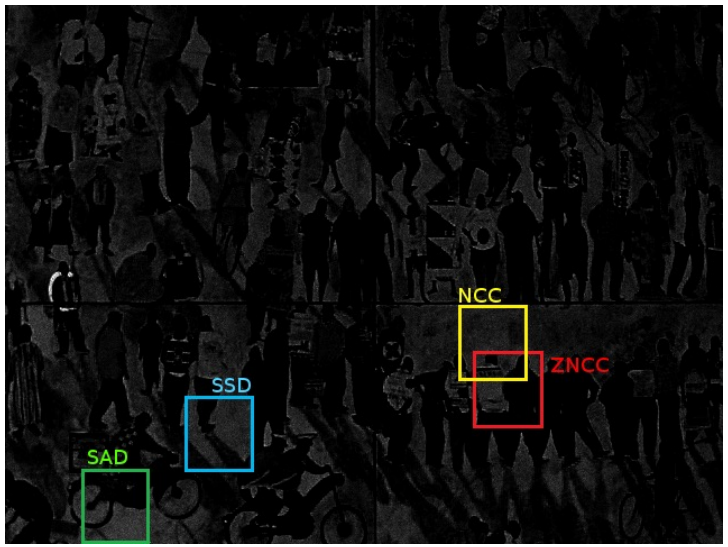
$$T(m,n) \rightarrow \left(T(m,n) - \mu(T)\right)$$

$$ZNCC(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - \mu(\tilde{I})\right)\cdot\left(T(m,n) - \mu(T)\right)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I(i+m, j+n) - \mu(\tilde{I})\right)^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(T(m,n) - \mu(T)\right)^2}}$$

- Invariant to affine intensity changes $\tilde{\mathbf{I}}(i,j) = \alpha\cdot\mathbf{T} + \beta$

Comparison under significant intensity changes





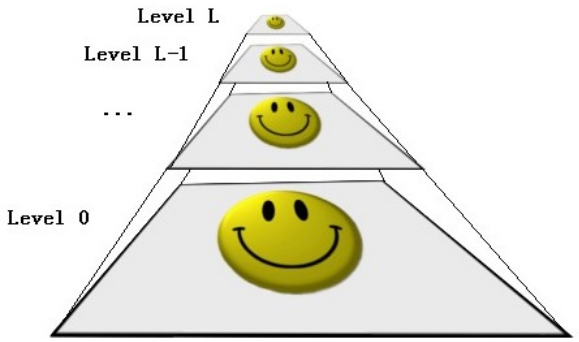
IT IS STILL EFFICIET WITH INTENSITY CHANGES
↳ JUST A LITTLE UNCERTERES

ZNCC turns out to be a similarity function very robust to intensity changes!

Fast Template Matching

- Template matching may be exceedingly slow whenever the model and/or target images have a large size (i.e. computational complexity is O(M×N×W×H))

- A popular approach is to deploy an image pyramid:
  - Smoothing and sub-sampling
    - Typically 1/2 on both sides at each level
    - Shrink the image, shrink the template
  - Full search at top level and then local refinements traversing back the pyramid down to the original image
  - Very fast approach, though the number of levels needs to be chosen carefully (and empirically) to avoid loss of information, which negatively affect the detection



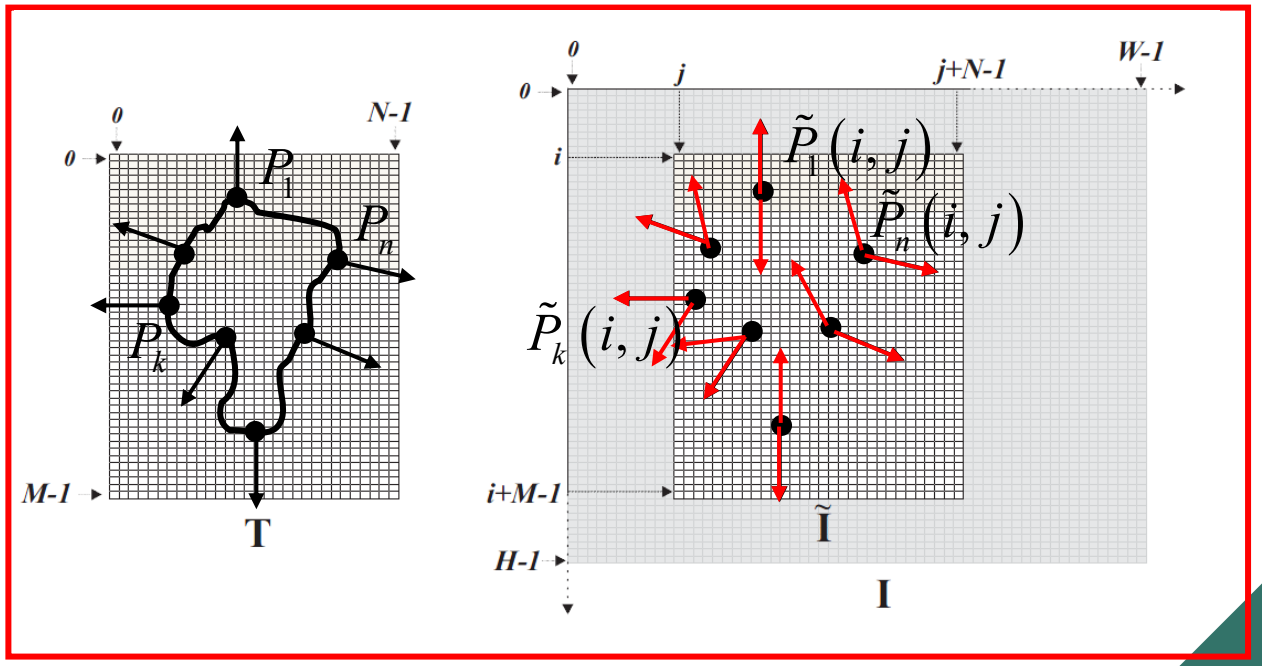
- Development of fast template matching algorithms is an active research topic

Shape-based Matching

- Edge-based template matching approach
  - First, a set of control points, P_k, is extracted from the model image by an Edge Detector and the gradient direction at each P_k is stored
    - Template composed by offsets and gradient directions
  - Then, at each position (i,j) of the target image, the **recorded** gradient directions associated with control points are compared to those at their corresponding image points, $P_k(i,j)$

- The more the red arrows are aligned to the black arrows the more similar the sub-image is to the template

- We do not perform edge detection on the target image, just on the template



Steger, 2002

Similarity Function

WE CARE ABOUT ORIENTATION AND NOT MAGNITUDE

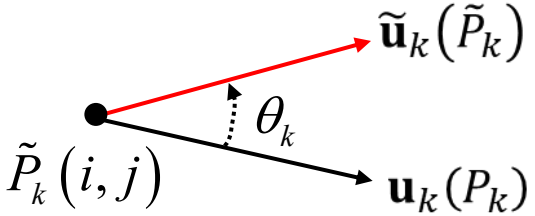
Normalized => unit vector

$$\mathbf{G}_k(P_k) = \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, \mathbf{u}_k(P_k) = \frac{1}{\|\mathbf{G}_k(P_k)\|} \begin{bmatrix} I_x(P_k) \\ I_y(P_k) \end{bmatrix}, k = 1..n$$ Template

$$\tilde{\mathbf{G}}_k(\tilde{P}_k) = \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{\|\tilde{\mathbf{G}}_k(\tilde{P}_k)\|} \begin{bmatrix} I_x(\tilde{P}_k) \\ I_y(\tilde{P}_k) \end{bmatrix}, k = 1..n$$ Target

$$S(i,j) = \frac{1}{n}\sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k) = \frac{1}{n}\sum_{k=1}^{n} \cos\theta_k$$

- The similarity function spans the interval [-1; 1]
  - It takes its maximum value when all the gradients at the control points in the current window of the target image are perfectly aligned to those at the control points of the model image
  - If they are perfeclty aligned the angle is zero, the cosine is 1, the sum is n, which divided by n is 1, and vice versa



- Choosing a detection threshold, S_{min}, can be thought of as specifying the fraction of model points which must be seen in the image to trigger a detection

More robust similarity functions

- Certain application settings call for invariance to *global inversion of contrast polarity* along object's contours, as the object may appear either darker or brighter than the background in the target image

- This kind of invariance can be achieved by a slight modification to the similarity function defined previously (global)

$$S(i,j) = \frac{1}{n}\left|\sum_{k=1}^{n} \mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k)\right| = \frac{1}{n}\left|\sum_{k=1}^{n} \cos\theta_k\right|$$

- The following function is even more robust due to the ability to withstand local contrast polarity inversions (local)

$$S(i,j) = \frac{1}{n}\sum_{k=1}^{n}\left|\mathbf{u}_k(P_k) \cdot \tilde{\mathbf{u}}_k(\tilde{P}_k)\right| \quad = \frac{1}{n}\sum_{k=1}^{n}|\cos\theta_k|$$

The Hough Transform

- The *Hough Transform (HT)* enables to detect objects having a known shape that *can be expressed by an equation* (e.g. lines, circles, ellipses..) based on projection of the input data into a suitable space referred to as *parameter* or *Hough space* (different from the image space)
- The HT turns a global detection problem into a local one (i.e., look for feature points into the parameter space, instead of looking for the whole shape in the image space)
- The HT is usually applied after an edge detection process (i.e., the actual input data consist of the edge pixels extracted from the original image)
- The HT is robust to noise and allows for detecting the sought shape even though it is partially occluded into the image (up to a certain user-selectable degree of occlusion)

- The HT was invented to detect lines and later extended to other analytical shapes (circle, ellipses) as well as to *arbitrary shapes => Generalized Hough Transform (GHT)*
- The GHT principle is widely deployed also within object detection pipelines relying on local invariant features such as, e.g., SIFT

Basic Principle

- HT formulation for lines, what is fixed and what is changing in this equqtion? $y - mx - c = 0$

- In the usual image space interpretation of the line equation the parameters $(\hat{m}, \hat{c})$ are fixed

$$y - \hat{m}x - \hat{c} = 0$$

so that the equation represents the mapping from point $(\hat{m}, \hat{c})$ of the *parameter space to the image points belonging to the line*

- However, we may instead fix $(\hat{x}, \hat{y})$

$$\hat{y} - m\hat{x} - c = 0$$

so the equation represents the mapping from image point $(\hat{x}, \hat{y})$ to the **parameter space providing all the lines through the image point**

Basic Principle

- Consider two image points P_1,P_2 and map both into the parameter space, we get two lines intersecting at the parameter space point representing the image line through P_1,P_2 :

$$\begin{cases} \hat{y}_1 - m\hat{x}_1 - c = 0 \\ \hat{y}_2 - m\hat{x}_2 - c = 0 \end{cases} \Longrightarrow \begin{cases} m = \frac{\hat{y}_2 - \hat{y}_1}{\hat{x}_2 - \hat{x}_1} \\ c = \frac{\hat{x}_2\hat{y}_1 - \hat{x}_1\hat{y}_2}{\hat{x}_2 - \hat{x}_1} \end{cases}$$

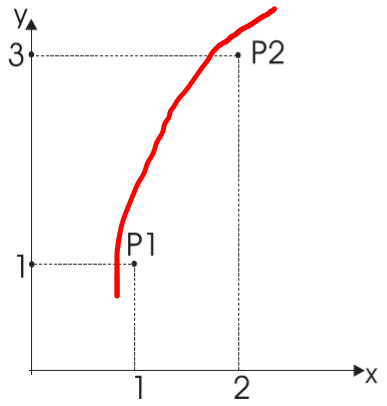
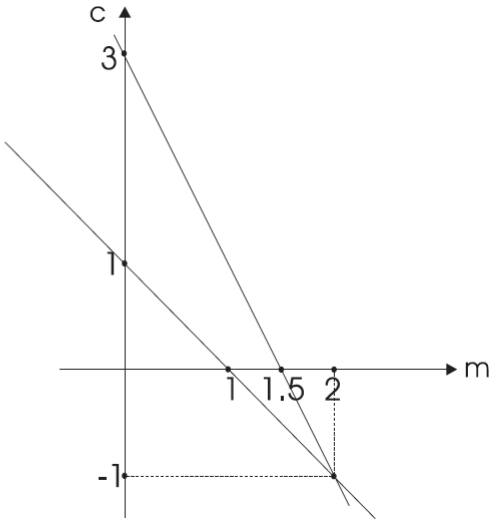


Image Space



Parameter Space

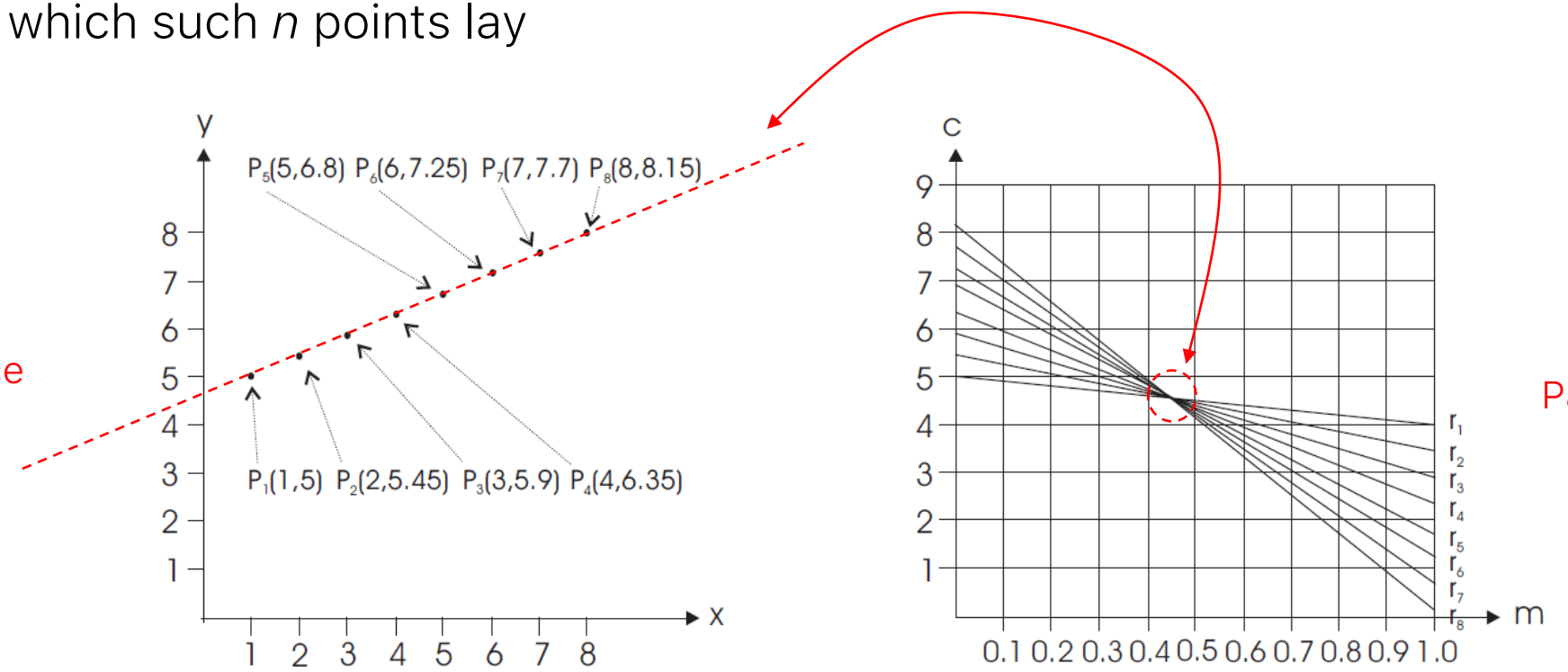
- *m* and *c*, represent the parameters of the line passing through P1 and P2

- More generally, if we map ***n*** image points we get as many intersections as ***n(n-1)/2*** (i.e. the number of lines through the n image points)

Basic Principle

- Considering *n* collinear image points, we can notice that their corresponding transforms (i.e. parameter space lines) will intersect at a single parameter space point representing the image line along which such *n* points lay

Image Space



Parameter Space

- Rather than looking at an extended shape into the image, we look for a specific feature (where lines intersect) in the parameter space of lines

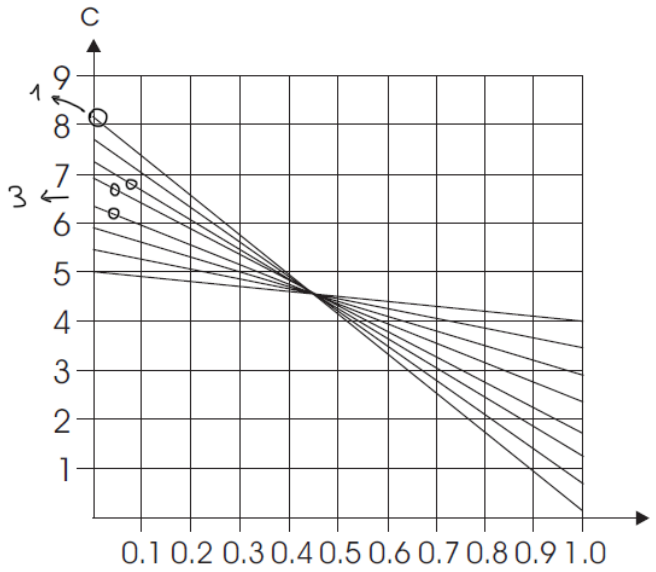
Basic Principle

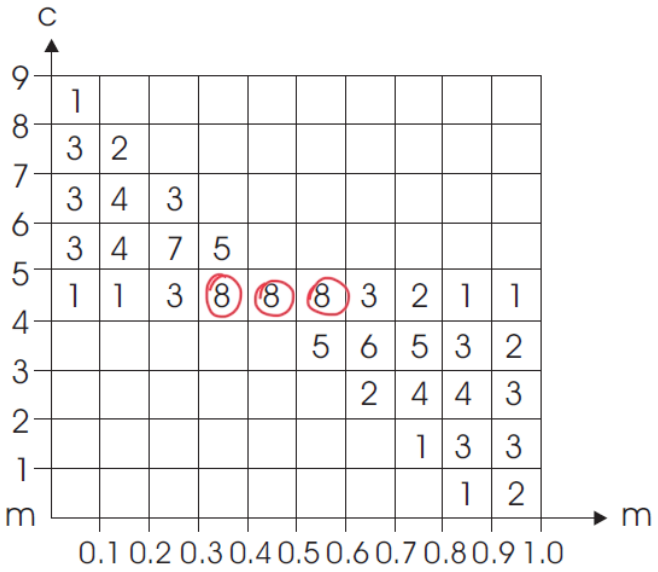
- Therefore, given a sought *analytic shape represented by a set of parameters*, the HT consists in mapping image points (i.e. usually edge points) so as to create *curves into the parameter space of the shape*
- *Intersections of parameter space curves* indicate the presence of image points explained *by a certain instance of the shape*
  - the more the intersecting *curves* the more are such image points and thus the higher is the evidence of the presence of that instance in the image
- Detecting objects through the HT consists in finding *parameter space points through which many curves do intersect* (a local rather than global detection problem)
- To make it work in practice, the parameter space needs to be quantized and allocated as a memory array, which is often refereed to as *Accumulator Array (AA)*
- Curves are "drawn" into the AA by a so called *voting process*:
  1. the transform equation is repeatedly computed to increment the bins satisfying the equation
  2. *a high number of intersecting curves at a point of the parameter space will provide a high number of votes* into a bin of the AA
  3. Finding parameter space points through which many curves do intersect is thus implemented in practice by finding *peaks of the AA*, i.e. local maxima showing a high number of votes

AA for line detection (Voting)

- The AA highlights the presence of a line with $m \in [0.3, 0.6],\ c \in [4, 5]$

Continuous Parameter Space



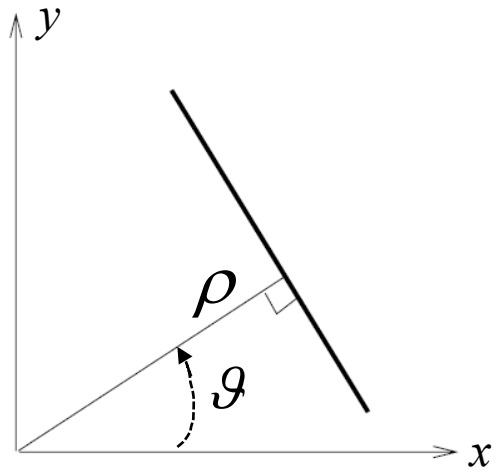


Quantized AA

- To detect the line more accurately, the AA should be quantized more finely → 3 max values {8}
- The HT is robust to noise because spurious votes due to noise unlikely accumulate into a bin so as to trigger a false detection
- A partially occluded object can be detected provided that the threshold on the minimum number of votes required to declare a detection is lowered according to the degree of occlusion to be handled

HT for Line Detection

- The usual line parametrization considered so far (i.e. *y-mx-c=0*) is impractical due to *m* spanning an infinite range
- The "normal parametrization" is adopted in the HT for lines: $\rho = x\cos\vartheta + y\sin\vartheta$



- Image points $(\hat{x}, \hat{y})$ are mapped into sinusoidal curves

  of the (ϑ, ρ) parameter space: $\rho = \hat{x}\cos\vartheta + \hat{y}\sin\vartheta$

- With the normal parametrization: $\vartheta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \quad \rho \in [-\rho_{\max}, \rho_{\max}]$

- while ρ_{max} is usually taken as large as the image diagonal: $N \times N \text{ pixels} \rightarrow \rho_{\max} = N \cdot \sqrt{2}$